DETERMINING OPERATIONAL STATUS OF A MOBILE DEVICE CAPABLE OF EXECUTING SERVER-SIDE APPLICATIONS

COPYRIGHT NOTICE

FIELD OF THE INVENTION

[0002] The present invention relates to software, devices and methods for determining the operational status of a mobile device capable of executing server-side applications.

BACKGROUND OF THE INVENTION

[0003] Wireless connectivity is a feature of the modern telecommunications environment. An increasing range of people are using a wide variety of wireless data networks to access corporate data applications.

[0004] However, there are numerous competing mobile devices (i.e. wireless computing devices) that can be used to achieve this. Each device has its own operating system and its own display characteristics. Operating systems are not mutually compatible, nor are the display characteristics--some are color, some are black and white, some are text-only, and some are pictorial.

[0005] At the same time, an increasing number of mobile device users are people without a technical background or high level of educational achievement.

Such people are often intimidated by the need to run complex installation programs. Furthermore, at present, such installation programs generally depend on cable connections to a personal computer by the means of a `cradle` or other such device.

[0006] U.S. Patent Publication No. US 2003/0060896, which is hereby incorporated by reference hereinto, discloses a mechanism allowing server-side applications to be presented at multiple wireless devices with minimal modification of the application at the server. As disclosed, the manner in which an application is presented at a mobile device is defined by a text based application definition file. The definition file describes how an application is to be presented at mobile device; the format of transactions over the wireless network; and a format of data related to the application to be stored at the mobile device. A virtual machine software component at the mobile device interprets the definition file and presents an interface to the application in accordance with the definition file. Conveniently, the application definition file may be independent of the particular type of mobile device, while virtual machine software components specific to the mobile device may be created.

[0007] The disclosed mechanism, while flexible, may have certain shortcomings. For example, in the event that an error occurs which interferes with normal application operation at the mobile device, which may manifest itself in the display of erroneous information at the mobile device or in the failure of the device to respond to stimuli, it may be difficult to ascertain whether the error is caused by the virtual machine software being used at the mobile device, a problem relating to the hardware at the mobile device (e.g. limited battery power or exhaustion of memory), or other problems. The user cannot not be relied upon to take steps to diagnose the problem at the mobile device because the user may lack the necessary technical expertise and because normal interaction with the device may be impossible.

[0008] Alternatively, it may simply be desirable to periodically assess the status of all or some of the mobile devices which are executing server-side applications, regardless of whether they are currently experiencing errors, e.g. in order to

compile a list of which errors (if any) have occurred most recently or most frequently at the mobile devices, to determine how the mobile devices are being used, or for other purposes.

[0009] A solution addressing at least some of the above-noted shortcomings would be desirable.

SUMMARY OF THE INVENTION

[0010] To determine the operational status of a wireless communication device capable of executing server-side applications, a message is sent to the device requesting operational status of the device. The message may be may be triggered by a system administrator at a middleware server. The wireless communication device may receive the message, compose a response message indicative of the operational status of the device, and send the response message back to the middleware server. The messages may be extensible markup language (XML) messages. Composition of the response message may entail verifying that a textual operational status description forming part of the response message omits illegal XML characters, e.g., by passing the description through an XML formatter for removal of any illegal XML characters.

[0011] In one aspect of the present invention, there is provided a method of determining operational status of a wireless communication device capable of executing server-side applications, the method comprising: sending a message to said wireless communication device capable of executing server-side applications requesting operational status of the device; and receiving a response message from said wireless communication device indicative of the operational status of the device.

[0012] In another aspect of the present invention, there is provided a method of providing the operational status of a wireless communication device capable of executing server-side applications, the method comprising: receiving a message at said wireless communication device capable of executing server-side applications

requesting operational status of the device, said receiving resulting in a received message; composing a response message from said wireless communication device indicative of the operational status of the device; and sending said response message from said wireless communication device to an originator of said received message.

[0013] In a further aspect of the present invention, there is provided a server comprising a processor and memory in communication with said processor storing machine-executable code adapting said server to: send a message to said wireless communication device capable of executing server-side applications requesting operational status of the device; and receive a response message from said wireless communication device indicative of the operational status of the device.

[0014] In a further aspect of the present invention, there is provided a wireless communication device comprising a processor and memory in communication with said processor storing machine-executable code adapting said device to: receive a message at said wireless communication device capable of executing server-side applications requesting operational status of the device, said receiving resulting in a received message; compose a response message from said wireless communication device indicative of the operational status of the device; and send said response message from said wireless communication device to an originator of said received message.

[0015] Other aspects and features of the present invention will become apparent to those of ordinary skill in the art upon review of the following description of specific embodiments of the invention in conjunction with the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

[0016] In the figures which illustrate by way of example only, embodiments of the present invention,

[0017] FIG. 1 schematically illustrates a mobile device, exemplary of an embodiment of the present invention, including virtual machine software, further exemplary of an embodiment of the present invention;

[0018] FIG. 2 further illustrates the organization of exemplary virtual machine software at the mobile device of FIG. 1;

[0019] FIG. 3 illustrates an operating environment for the device of FIG. 1;

[0020] FIG. 4 illustrates the structure of example application definitions stored at a server of FIG. 2 used by the device of FIG. 1;

[0021] FIG. 5 schematically illustrates the formation of application definition files at a middleware server of FIG. 2;

[0022] FIG. 6 schematically illustrates the middleware server of FIG. 2, exemplary of an embodiment of the present invention, including an application definitions database, further exemplary of an embodiment of the present invention;

[0023] FIG. 7 is a flow diagram illustrating the exchange of sample messages passed between a mobile device, middleware server and application server of FIG. 2;

[0024] FIGS. 8-11 illustrate steps performed at a mobile device under control of virtual machine software of FIG. 2;

[0025] FIG. 12 illustrates the format of messages exchanged in the message flow of FIG. 7;

[0026] FIG. 13 illustrates the format of a simple ping message which may be sent to a mobile device by a middleware server;

[0027] FIG. 14 illustrates the format of a response to a simple ping message which may be sent by a mobile device to a middleware server;

[0028] FIG. 15 illustrates the format of a device statistics ping message which may be sent to a mobile device by a middleware server;

[0029] FIG. 16 illustrates the format of a response to a device statistics ping message which may be sent by a mobile device to a middleware server;

[0030] FIGS. 17A-17PPP contain Appendix "A" detailing example eXtensible Markup Language (XML) entities understood by the virtual machine software of the mobile device of FIG. 1.

DETAILED DESCRIPTION

[0031] FIG. 1 illustrates a mobile device 10, exemplary of an embodiment of the present invention. Mobile device 10 may be any conventional mobile device, modified to function in manners exemplary of the present invention. As such, mobile device 10 includes a processor 12, in communication with a network interface 14, storage memory 16, and a user interface 18 typically including a keypad and/or touch-screen. Network interface 14 enables device 10 to transmit and receive data over a wireless network 22. Mobile device 10 may be, for example, be a Research in Motion (RIM) two-way paging device, a WinCE based device, a PalmOS device, a WAP enabled mobile telephone, or the like. Memory 16 of device 10 stores a mobile operating system such as the PalmOS, or WinCE operating system software 20. Operating system software 20 typically includes graphical user interface and network interface software having suitable application programmer interfaces ("API"s) for use by other applications executing at device 10.

[0032] Memory at device 10 further stores virtual machine software 24, exemplary of an embodiment of the present invention. Virtual machine software 24, when executed by mobile device 10, enables device 10 to present an interface for server side applications provided by a middleware server, described below. Specifically, virtual machine software 24 interprets a text application definition file defining a user interface 18 controlling application functionality, and the display format (including display flow) at device 10 for a particular server-side application; the format of data to be exchanged over the wireless network for the application; and the format of data to be stored locally at device 10 for the application. Virtual

machine software 24 uses operating system 20 and associated APIs to interact with device 10, in accordance with the received application definition file. In this way, device 10 may present interfaces for a variety of applications, stored at a server. From the perspective of operating system 20, virtual machine software 24 is viewed as another application resident at device 10. Moreover, multiple wireless devices each having a similar virtual machine software 24 may use a common server side application in combination with an application definition, to present a user interface and program flow specifically adapted for the device.

[0033] As such, and as will become apparent, the exemplary virtual machine software 24 is specifically adapted to work with the particular mobile device 10. Thus if device 10 is a RIM Blackberry device, virtual machine software 24 is a RIM virtual machine. Similarly, if device 10 is a PalmOS or WinCE device, virtual machine software 24 would be a PalmOS or a WinCE virtual machine. As further illustrated in FIG. 1, virtual machine software 24 is capable of accessing local storage 26 at device 10.

[0034] As detailed below, an exemplary application definition file may be formed using a mark-up language, like XML. In accordance with an embodiment of the present invention, defined XML entities are understood by the virtual machine software 24. Defined XML entities are detailed in Appendix "A" hereto (the AIRIX Markup Language (ARML) Specification) and Appendix “A” of U.S. Patent Publication No. 2003/0060896. ARML is an XML markup language used in the present embodiment. The defined XML entities are interpreted by the virtual machine software 24, and may be used as building blocks to present server-side applications at mobile device 10, as detailed herein.

[0035] Specifically, as illustrated in FIG. 2, virtual machine software 24 includes a conventional XML parser 61; an event handler 65; a screen generation engine 67; and object classes 69 corresponding to XML entities supported by the virtual machine software 24, and possibly contained within an application definition file 28. Supported XML entities are detailed in Appendix "A" hereto. A person of ordinary skill will readily appreciate that those XML entities identified in Appendix "A" are exemplary only, and may be extended, or shortened as desired.

[0036] XML parser 61 may be formed in accordance with the Document Object Model, or DOM, available at www.w3.org/DOM/ and being incorporated by reference hereinto. Parser 61 enables virtual machine software 24 to read an application definition file. Using the parser, the virtual machine software 24 may form a binary representation of the application definition file for storage at the mobile device, thereby eliminating the need to parse text each time an application is used. Parser 61 may convert each XML tag contained in the application definition file, and its associated data to tokens, for later processing. As will become apparent, this may avoid the need to repeatedly parse the text of an application definition file.

[0037] Screen generation engine 67 displays initial and subsequent screens at the mobile device, in accordance with an application definition 28, as detailed below.

[0038] Event handler 65, of virtual machine software 24 allows device 10 under control of virtual machine software 24 to react to certain external events. Example events include user interaction with presented screens or display elements, incoming messages received from a wireless network, or the like.

[0039] Object classes 69 also form part of virtual machine 24 and define objects that allow device 10 to process each of the supported XML entities at the mobile device. Each of object classes 69 includes attributes used to store parameters defined by the XML file, and functions allowing the XML entity to be processed at the mobile device, as detailed in Appendix "A", for each supported XML entity. So, as should be apparent, supported XML entities are extensible. Virtual machine software 24 may be expanded to support XML entities not detailed in Appendix "A". Corresponding object classes could be added to virtual machine software 24.

[0040] As detailed below, upon invocation of a particular application at mobile device 10, the virtual machine software 24 presents an initial screen based on the contents of the application definition 28 for the application. Screen elements are created by screen generation engine 67 by creating instances of corresponding object classes for defined elements, as contained within object classes 69. The object instances are created using attributes contained in the application definition

file 28. Thereafter the event handler 65 of the virtual machine software 24 reacts to events for the application. Again, the event handler consults the contents of the application definition file for the application in order to properly react to events. Events may be reacted to by creating instances of associated "action" objects, from object classes 69 of virtual machine software 24.

[0041] Similarly, object classes 69 of virtual machine software 24 further include object classes corresponding to data tables and network transactions defined in the Table Definition and Package Definition sections of Appendix "A". At run time, instances of object classes corresponding to these classes are created and populated with parameters contained within application definition file, as required.

[0042] Using this general description, persons of ordinary skill in the art will be able to form virtual machine software 24 for any particular device. Typically, virtual machine software 24 may be formed using conventional object oriented programming techniques, and existing device libraries and APIs, as to function as detailed herein. As will be appreciated, the particular format of screen generation engine 67, object classes 69 will vary depending on the type of virtual machine software, its operating system and API available at the device. Once formed, a machine executable version of virtual machine software 24 may be loaded and stored at a mobile device, using conventional techniques. It can be embedded in ROM, loaded into RAM over a network, or from a computer readable medium.

[0043] Although, in the preferred embodiment the virtual machine software 24 and software forming object classes 29 are formed using object oriented structures, persons of ordinary skill will readily appreciate that other approaches could be used to form suitable virtual machine software. For example, object classes 69 forming part of the virtual machine could be replaced by equivalent functions, data structures or subroutines formed using a conventional (i.e. non-object oriented) programming environment. Operation of virtual machine software 24 under control of an application definition containing various XML definitions exemplified in Appendix "A" is further detailed below.

[0044] FIG. 3 illustrates the operating environment for a mobile device 10. Further example mobile devices 30, 32 and 34 are also illustrated in FIG. 3. These

mobile devices 30, 32 and 34 are similar to device 10 and also store and execute virtual machine software exemplary of an embodiment of the present invention.

[0045] Virtual machine software like that stored at device 10, executes on each mobile device 10, 30, 32, 34, and communicates with a middleware server 44 by way of example wireless networks 36 and 38 and network gateways 40 and 42. Example gateways 40 and 42 are generally available as a service for those people wishing to have data access to wireless networks. Wireless networks 36 and 38 are further connected to one or more computer data networks, such as the Internet and/or private data networks by way of gateway 40 or 42. As will be appreciated, the invention may work with many types of wireless networks. Middleware server 44 is in turn in communication with a data network, that is in communication with wireless network 36 and 38. The communication used for such communication is via TCP/IP over an HTTP transport. As could be appreciated, other network protocols such as X.25 or SNA could equally be used for this purpose.

[0046] At least three categories of communication between middleware server 44 and mobile devices 10, 30, 32 and 34 exist. First, virtual machine software 24 at each device may query middleware server 44 for a list of applications that a user of an associated mobile device 10, 30, 32 or 34 can make use of. If a user decides to use a particular application, device 10, 30, 32 or 34 can download a text description, in the form of an application definition file, for the application from the middleware server 44 over its wireless interface. As noted, the text description is preferably formatted using XML. Second, virtual machine software 24 may send, receive, present, and locally store data related to the execution of applications, or its own internal operations. The format of exchanged data for each application is defined by an associated application definition file. Again, the exchanged data is formatted using XML, in accordance with the application definition file. Third, middleware server 44 may query a mobile device as to its operational status, either for purposes of diagnosing a reported problem at the device or simply to collect mobile device operational status statistics. As will become apparent, it is the third category of communication which is the focus of the present application.

[0047] Middleware server 44 stores text application definition files for those

applications that have been enabled to work with the various devices 10, 30, 32, and 34 using virtual machine software 24 in a pre-defined format understood by virtual machine software 24. Software providing the functions of the middleware server 44, in the exemplary embodiment is written in C#, using SQL Server or MySQL database.

[0048] As noted, text files defining application definitions and data may be formatted in XML. For example XML version 1.0, detailed in the XML version 1.0 specification third edition and available at www.w3.org/TR/2004/REC-xml-20040404, may be used. However, as will be appreciated by those of ordinary skill in the art, the functionality of storing XML description files is not dependent on the use of any given programming language or database system.

[0049] Each application definition file is formatted according to defined rules and uses pre-determined XML mark-up tags known by both virtual machine software 24, and complementary middleware server software 68. That is, each application definition file 26 is an XML data instance file which conforms to a predefined XML schema designed to support the execution of server-side applications at various types of mobile devices. Tags define XML entities used as building blocks to present an application at a mobile device. Knowledge of these rules, and an understanding of how each tag and section of text should be interpreted, allows virtual machine software 24 to process an XML application definition and thereafter execute an application, as described below. Virtual machine software 24 effectively acts as an interpreter for a given application definition file.

[0050] FIG. 4 illustrates an example format for an XML application definition file 28. As illustrated, the example application definition file 28 for a given device and application includes three components: a user interface definition section 48, specific to the user interface for the device 10, which defines the format of screen or screens for the application and how the user interacts with them and contains application flow control events and actions; a network transactions definition section 50 defining the format of data to be exchanged with the application; and a local data definition section 52 defining the format of data to be stored locally on the mobile device by the application.

[0051] Defined XML mark-up tags correspond to XML entities supported at a device, and are used to create an application definition file 28. The defined tags may broadly be classified into three categories, corresponding to the three sections 48, 50 and 52 of an application definition file 28.

[0052] Example XML tags and their corresponding significance are detailed in Appendix "A". As noted above, virtual machine software 24 at a mobile device includes object classes corresponding to each of the XML tags. At run time, instances of the objects are created as required.

[0053] Broadly, the following list includes example XML tags (i.e. XML elements) which may be used to define the user interface definition:

SCREEN -- this defines a screen. A SCREEN tag pair contains the definitions of the user interface elements (buttons, radio buttons, and the like) and the events associated with the screen and its elements

BUTTON -- this tag defines a button and its associated attributes

LIST -- this tag defines a list box

CHOICEBOX -- this tag defines a choice item, that allows selection of a value from predefined list

MENU -- the application developer will use this tag to define a menu for a given screen

EDITBOX -- this tag defines an edit box

TEXT ITEM -- this tag describes a text label that is displayed

CHECKBOX -- this tag describes a checkbox

HELP -- this tag can define a help topic that is used by another element on

the screen

IMAGE -- this tag describes an image that appears on those displays that support images

ICON -- this tag describes an icon

EVENT -- this defines an event to be processed by the virtual machine software. Events can be defined against the application as a whole, individual screens or individual items on a given screen. Sample events would be receipt of data over the wireless interface, or a edit of text in an edit box

ACTION -- this describes a particular action that might be associated with an event handler. Sample actions would be navigating to a new window or displaying a message box.

[0054] The second category of example XML tags describes the network transaction section 50 of application definition 28. These may include the following example XML tags:

TABLEUPDATE--using this tag, the application developer can define an update that is performed to a table in the device's local storage. Attributes allow the update to be performed against multiple rows in a given table at once;

PACKAGEFIELD--this tag is used to define a field in a data package that passes over the wireless interface

[0055] The third category of XML tags used to describe an application are those used to define a logical database that may be stored at the mobile device. The tags available that may be used in this section are:

TABLE--this tag, and its attributes, define a table. Contained within a pair of TABLE tags are definitions of the fields contained in that table. The attributes

of a table control such standard relational database functions as the primary key for the table.

FIELD--this tag describes a field and its attributes. Attributes of a field are those found in a standard relational database system, such as the data type, whether the field relates to one in a different table, the need to index the field, and so on.

[0056] In addition to these XML tags, virtual machine software 24 may, from time to time, need to perform certain administrative functions on behalf of a user. In order to do this, one of object classes 69 has its own repertoire of tags to intercommunicate with the middleware server 44. Such tags differ from the previous three groupings in that they do not form part of an application definition file, but are solely used for administrative communications between the virtual machine software 24 and the middleware server 44. Data packages using these tags are composed and sent due to user interactions with the virtual machine's configuration screens. The tags used for this include:

REG--this allows the application to register and deregister a user for use with the middleware server

FINDAPPS--by using this operation, users can interrogate the server for the list of applications that are available to them

APP REG--using this operation, the end-user can register (or deregister) for an application and have the application interface downloaded automatically to their device (or remove the interface description from the device's local storage).

SETACTIVE--If the user's preferred device is malfunctioning, or out of power or coverage, they will need a mechanism to tell the Server to attempt delivery to a different device. The SETACTIVE command allows the user to set the device that they are currently using as their active one

PING--this XML element is used by the middleware server 44 to query a mobile device as to its operational status. It is described in greater detail

later in the application.

[0057] Referring again generally to the manner in which execution of server-based applications at mobile devices is facilitated, FIG. 5 illustrates the organization of application definitions at middleware server 44 and how middleware server 44 may form an application definition file 28 (FIG. 4) for a given device 10, 30, 32 or 34. In the illustration of FIG. 5, only two mobile devices 10 and 30 are considered. Typically, since network transactions and local data are the same across devices, the only piece of the application definition that varies for different devices is the user interface definition.

[0058] So, middleware server 44 stores a master definition 58 for a given server side application. This master definition 58 contains example user interface descriptions 48, 54, 56 for each possible type of mobile device 10, 30, 32; descriptions of the network transactions 50 that are possible and data descriptions 52 of the data to be stored locally on the mobile device. Preferably, the network transactions 50 and data descriptions 52 will be the same for all mobile devices 10, 30 and 32.

[0059] For device 10, middleware server 44 composes an application definition file 28 by querying the device type and adding an appropriate user interface description 48 for device 10 to the definitions for the network transactions 50 and the data 52. For device 30, middleware server 44 composes the application definition by adding the user interface description 54 for device 10 to the definitions for the network transactions 50 and data 52.

[0060] The master definition 58 for a given application is created away from the middleware server and loaded onto the middleware server by administrative staff charged with its operation. Master definition files could be created either by use of a simple text editor, or by a graphical file generation tool. Such a tool might generate part or all of the file, using knowledge of the XML formatting rules, based on the user's interaction with screen painters, graphical data definition tools and the like. It will be appreciated that the master definition file 58 is an XML data instance which conforms to a predefined XML schema referenced above

[0061] FIG. 6 illustrates the organization of middleware server 44 and associated master definitions. Middleware server 44 may be any conventional application server, modified to function in manners exemplary of the present invention. As such, middleware server 44 includes a processor 60, in communication with a network interface 66 and storage memory 64. Middleware server 44 may be, for example, a Windows 2000 server, a Sun Solaris server, or the like. Memory of middleware server 44 stores an operating system such as Windows 2000, or Solaris operating system software 62.

[0062] Network interface 66 enables middleware server 44 to transmit and receive data over a data network 63. Transmissions are used to communicate with both the virtual machine software 24 (via the wireless networks 36, 38 and wireless gateways 40,42) and one or more application servers, such as application server 70, that are the end recipients of data sent from the mobile client applications and the generators of data that is sent to the mobile client applications.

[0063] Memory at middleware server 44 further stores software 68, exemplary of an embodiment of the present invention. Middleware server software 68, when executed by middleware server 44 enables the middleware server to understand and compose XML data packages that are sent and received by the middleware server. These packages may be exchanged between middleware server 44 and the virtual machine software 24, or between the middleware server 44 and the application server 70. Middleware server software 68 may be loaded from a machine-readable medium.

[0064] As described above, communication between the application server 70 and the middleware server 44 can, in an exemplary embodiment, use HTTP running on top of a standard TCP/IP stack; however this is not a requirement. An HTTP connection between a running application at the application server 70 and the middleware server 44 is established in response to the application at a mobile device presenting the application. The server side application provides output to middleware server 44 over this connection. The server side application data is formatted into appropriate XML data packages understood by the virtual machine software 24 at a mobile device by the server side application.

[0065] That is, a server side application (or an interface portion of the application) formats application output into XML in a manner consistent with the format defined by the application definition file for the application. Alternatively, an interface component separate from the application could easily be formed with an understanding of the format and output for a particular application. That is, with a knowledge of the format of data provided and expected by an application at application server 70, an interface component could be a produced using techniques readily understood by those of ordinary skill. The interface portion could translate application output to XML, as expected by middleware server 44. Similarly, the interface portion may translate XML input from a mobile device into a format understood by the server side application.

[0066] The particular identity of the mobile device on which the application is to be presented may be identified by a suitable identifier, in the form of a header contained in the server side application output. This header may be used by middleware server 44 to forward the data to the appropriate mobile device. Alternatively, the identity of the connection could be used to forward the data to the appropriate mobile device.

[0067] FIG. 7 illustrates a sequence diagram detailing data (application data or application definition files 28) flow between mobile device 10 and middleware server 44, in manners exemplary of an embodiment of the present invention.

[0068] For data requested from middleware server 44, device 10, under software control by virtual machine software 24 makes requests to middleware server 44 (also illustrated in FIG. 2), which passes over the wireless network 36 through network gateway 40. Network gateway 40 passes the request to the middleware server 44. Middleware server 44 responds by executing a database query on its database 46 that finds which applications are available to the user and the user's mobile device. For data passed from middleware server 44 to device 10, data is routed through network gateway 40. Network gateway 40 forwards the information to the user's mobile device over the wireless network 36.

[0069] FIG. 7 when considered with FIG. 3 illustrates a sequence of communications between device 10, and middleware server 44 that may occur

when the user of a mobile device wishes to download an application definition file 28 for a server side application.

[0070] So, initially, device 10 interrogates server 44 to determine which applications are available for the particular mobile device being used. This may be accomplished by the user instructing the virtual machine software 24 at device 10 to interrogate the server 44. Responsive to these instructions the virtual machine software 24 sends an XML message to the server requesting the list of applications (data flow 72); as illustrated in FIG. 7 the XML message may contain the <FINDAPPS> tag, signifying to the middleware server 44, its desire for a list available application. In response, middleware server 44 makes a query to database 46. Database 46, responsive to this query, returns a list of applications that are available to the user and the mobile device. The list is typically based, at least in part, on the type of mobile device making the request, and the applications known to middleware server 44. Middleware server 44 converts this list to an XML message and sends to the virtual machine (data flow 74). Again, a suitable XML tag identifies the message as containing the list of available applications.

[0071] In response, a user at device 10 may choose to register for an available server side application. When a user chooses to register for an application, virtual machine software 24 at device 10 composes and sends an XML registration request for a selected application (data flow 76) to middleware server 44. As illustrated in FIG. 11, an XML message containing a <REG> tag is sent to middleware server 44. The name of the application is specified in the message. The middleware server 44, in response, queries its database for the user interface definition for the selected application for the user's mobile device. Thereafter, the middleware server creates the application definition file, as detailed with reference to FIG. 5. Then, middleware server 44 sends to the mobile device (data flow 78 – FIG. 7) the created application definition file 28.

[0072] The user is then able to use the functionality defined by the interface description to send and receive data.

[0073] At this time, parser 61 of virtual machine software 24 may parse the XML text of the application definition file to form a tokenized version of the file. That is,

each XML tag may be converted a defined token for compact storage, and to minimize repeated parsing of the XML text file. The tokenized version of the application definition file may be stored for immediate or later use by device 10. In this context, the term "tokenized" may refer to placement of the XML structure into binary objects, which is much like conversion of a script into byte code.

[0074] Thereafter, upon invocation of a particular application for which the device 10 has registered, the screen generation engine 67 of the virtual machine software 24 at the device causes the virtual device to locate the definition of an initial screen for that application. The initial screen is identified within the application definition file 28 for that application using a <SCREEN> tag, and an associated attribute of <First screen="yes">.

[0075] Steps performed by virtual machine software 24 in processing a first or subsequent screen are illustrated in FIG. 8. As illustrated, screen generation engine 67, generates an instance of an object class, defining a screen by parsing the section of the XML application definition file corresponding to the <SCREEN> tag in step S802. Supported screen elements may be buttons, edit boxes, menus, list boxes, and choice items, as identified in Appendix "A". Other screen elements, such as images and checkboxes, as detailed in Appendix "A" may also be supported. For clarity of illustration, their processing by screen generation engine 67 however, is not detailed. Each supported tag under the SCREEN definition section, in turn causes creation of instances of object classes within the virtual machine software 24. Typically, instances of objects corresponding to the tags, used for creation of a screen, result in presentation of data at mobile device 10. As well the creation of such objects may give rise to events (e.g. user interaction) and actions to be processed at device 10.

[0076] Each element definition causes virtual machine software 24 to use the operating system of the mobile device to create corresponding display element of a graphical user interface as more particularly illustrated in FIG. 9. Specifically, for each element, the associated XML definition is read in step S806, S816, S826, S836, and S846, and a corresponding instance of a screen object defined as part of the virtual machine software 24 is created by the virtual machine software 24 in

steps S808, S818, S828, S838 and S848, in accordance with steps S902 and onward illustrated in FIG. 9. Each interface object instance is created in step S902. Each instance takes as attribute values defined by the XML text associated with the element. A method of the virtual machine object is further called in step S904, and causes a corresponding device operating system object to be created. Those attributes defined in the XML text file, stored within the virtual machine object instance are applied to the corresponding instance of a display object created using the device operating system in steps S908S-S914. These steps are repeated for all attributes of the virtual machine object instance. For any element allowing user interaction, giving rise to an operating system event, the event handler 65 of virtual machine software 24 is registered to process operating system events, as detailed below.

[0077] Additionally, for each event (as identified by an <EVENT> tag) and action (as identified by an <ACTION> tag) associated with each XML element, virtual machine software 24 creates an instance of a corresponding event and action object forming part of virtual machine software 24. Virtual machine software 24 further maintains a list identifying each instance of each event and action object, and an associated identifier of an event in steps S916 to S928.

[0078] Steps S902-S930 are repeated for each element of the screen in steps S808, S818, S828, S838 and S848 as illustrated in FIG. 8. All elements between the <SCREEN> definition tags are so processed. After the entire screen has been so created in memory, it is displayed in step S854, using conventional techniques.

[0079] As will be appreciated, objects are specific to the type of device executing the virtual machine software 24. Functions initiated as a result of the XML description may require event handling. This event handling is processed by event handler 65 of virtual machine software 24 in accordance with the application definition file 28. Similarly, receipt of data from a mobile network will give rise to events. Event handler 65, associated with a particular application presented at the device similarly processes incoming messages for that particular application. In response to the events, virtual machine software 24 creates instance of software objects, and calls functions of those object instances, as required by the definitions

contained within the XML definitions contained within the application definition file 28, giving rise to the event.

[0080] As noted, the virtual machine software 24 includes object classes, allowing the virtual machine to create object instances corresponding to an <EVENT> tag. The event object classes includes methods specific to the mobile device that allow the device to process each of the defined XML descriptions contained within the application definition file, and also to process program/event flow resulting from the processing of each XML description.

[0081] Events may be handled by virtual machine software 24 as illustrated in FIG. 10. Specifically, as device handler 65 has been registered with the operating system for created objects, upon occurrence of an event, steps S1002 and onward are performed in response to the operating system detecting an event.

[0082] An identifier of the event is passed to event handler 65 in step S1002. In steps S1004-S1008, this identifier is compared to the known list of events, created as a result of steps S916-S930. For an identified event, actions associated with that event are processed in step S1008-S1014.

[0083] That is, virtual machine software 24 performs the action defined as a result of the <ACTION> tag associated with the <EVENT> tag corresponding to the event giving rise to processing by the event handler 65. The <ACTION> may cause creation of a new screen, as defined by a screen tag, a network transmission, a local storage, or the like.

[0084] New screens, in turn, are created by invocation of the screen generation engine 61, as detailed in FIGS. 8 and 9. In this manner the navigation through the screens of the application is accomplished according to the definition embodied in the XML application definition.

[0085] Similarly, when the user wishes to communicate with the middleware server, or store data locally, event handler 65 creates instances of corresponding object classes within the object classes 69 of virtual machine software 24 and calls their methods to store or transmit the data using the local device operating system.

The format of data is defined by the device local definition section 52; the format of network packages is defined in the network transaction package definition section 50.

[0086] For example, data that is to be sent to the wireless network is assembled into the correct XML packages using methods within an XML builder object, formed as a result of creating an instance of a corresponding object class within object classes 69 of virtual machine software 24. Methods of the XML builder object create a full XML package before passing the completed XML package to another message server object. The message server object uses the device's network APIs to transmits the assembled data package across the wireless network.

[0087] Received XML data packages from network 63 (FIG. 2) give rise to events processed by event handler 65. Processing of the receipt of data packages is not specifically illustrated in FIG. 9. However, the receipt of data triggers a "data" event of the mobile device's operating system. This data event is passed to the virtual machine, and event handler 65 inspects the package received. As long as the data received is a valid XML data package as contained within the application definition, the virtual machine inspects the list of recognised XML entities.

[0088] So, for example, a user could send a login request 80 by interacting with an initial login screen, defined in the application definition file for the application. This would be passed by the middleware server 44 to the backend application server 70. The backend application server according to the logic embedded within its application, would return a response, which the middleware server 44 would pass to the virtual machine software 24. Other applications, running on the same or other application servers might involve different interactions, the nature of such interactions being based upon the functionality and logic embedded within the application server 70.

[0089] FIG. 12 illustrates sample XML messages passed as the result of message flows illustrated in FIG. 6. For each message, the header portion, between the <HEAD> . . . </HEAD> tags contains a timestamp and the identifier of the sending device.

[0090] Example message 72 is sent by the mobile device to request the list of applications that the server has available to that user on that device. It specifies the type of device by a text ID contained between the <PLATFORM> . . . </PLATFORM> tags. Example message 74 is sent in response to message 70 by middleware server 44 to the mobile device 10. It contains a set of <APP> . . . </APP> tag pairs, each of which identifying a single application that is available to the user at device 10. Example message 76 is sent from the mobile device 10 to middleware server 44 to register for a single server side application. The tags specify information about the user. Message 78 is sent by the middleware server 44 to the mobile device in response to a request to register device 10 for an application. The pair of tags <VALUE> . . . </VALUE> gives a code indicating success or failure. In the sample message shown, a success is shown, and is followed by the interface description for the application, contained between the <INTERFACE> . . . </INTERFACE> tags. This interface description may then be stored locally within memory 16 of device 10.

[0091] As noted, when a user starts an application that has been downloaded in the manner described above, the virtual machine software 24 reads the interface description that was downloaded for that device 10, and the virtual machine software 24 identifies the screen that should be displayed on startup, and displays its elements as detailed in relation to FIGS. 9 and 10. The user may then use the functionality defined by the user interface definition section 48 of the application definition 28 to send and receive data from a server side application.

[0092] In the event that an error should occur which interferes with normal application operation at a mobile device such as mobile device 10, which error may manifest itself in the display of erroneous information at the mobile device or in a failure of the mobile device to respond to stimuli for example, a user at the mobile device will likely wish to ascertain the cause of the problem with a view to rectifying the problem. In this situation, the user of the mobile device may notify a system administrator at the middleware server 44 (e.g. by telephone or email from a different computing device) so that the administrator may take steps to assess the operational status of the mobile device on the user's behalf. Alternatively, the system administrator may simply wish to assess the status of all or some of the

mobile devices which are executing server-side applications, regardless of whether they are currently experiencing errors, for statistical purposes.

[0093] In either case, the system administrator may interact with the middleware server software 68 so as to cause the middleware server to "ping" the relevant mobile device to solicit operational status information from the device. Two types of "pings" may exist in an exemplary embodiment. The first type of "ping" is simply an "are you alive" query which either results in a response in the affirmative (if the device is active) or no response (if the device is inactive). The second type of "ping" is a request for device statistics which results in a response containing various forms of device statistics from the relevant mobile device, as described in greater detail below, or no response (if the device is inactive).

[0094] For the first type of ping, a message having the format 1300 illustrated in FIG. 13 may be generated and enqueued for transmission to the mobile device. The message priority may be set sufficiently high so that the message will be sent before all other messages that are queued for transmission to the device. The message may be received by the device through normal device push or pull, depending on how the device is configured.

[0095] The administrator's actions may initially cause a new database table TBLMOBILEREQUESTS to be created at the middleware server 44, with the following fields:

lngMobileID integer,

dtmLastPingRequest datetime,

dtmLastPingResponse datetime,

dtmLastStatisticsRequest datetime,

dtmLastStatisticsResponse datetime,

varStatisticsResponse memo

[0096] If there is no record for the relevant mobile device having the MobileID, one may be inserted into the table, initialized with the current time (e.g. in UTC) in

dtmLastPingRequest. The rest of the fields may be null. If there is a record for the current MobileID, dtmLastPingRequest may be set to the current time (e.g. in UTC), and dtmLastPingResponse will be set to null. The other fields need not be updated.

[0097] If the record in TBLMOBILEREQUESTS cannot be created/updated for some reason, it may be desirable to notify the system administrator at middleware server 44 of same, e.g., via a message box, and to abort the process without enqueueing a message for transmission. If the message cannot be queued for any reason, it may be desirable to notify the administrator that the ping could not be queued.

[0098] The middleware server 44 handles the response to the ping message from the mobile device. The server 44 may stamp the time the response is received into the ping record.

[0099] When the ping response is received, the database table TBLMOBILEREQUESTS may be updated for the responding MobileID. The field dtmLastPingResponse may be set to the current date and time (e.g. in UTC). If the record in TBLMOBILEREQUESTS cannot be updated, an error may be written to an error log. Administrator notification of the receipt of a ping response does not necessarily occur automatically.

[00100] The ping response may be viewable by the system administrator using the middleware server software 68. The dtmLastPingRequest and dtmLastPingResponse fields may be read from the table TBLMOBILEREQUESTS. A screen may be displayed (e.g. may pop up), containing these two date/times, formatted using the current time zone and locale information. The screen may contain a "Refresh" button, which will retrieve the current values from the table.

[00101] If there is no record for the current mobile device, or it cannot be retrieved, the fields may be displayed as being empty.

[00102] The system administrator may be required to manually refresh the screen until the ping response is retrieved. However, the administrator may be able

to watch an outgoing queue for that device to determine when the device receives the ping.

[00103] At the mobile device side, when a "simple" ping request has been received, the device immediately formats, queues, and sends a ping response, which may have the format 1400 of FIG. 14, indicating simply that the device is "alive". The response priority may be set to a high priority, so that the response will be sent ahead of any other messages. If no entry can be created in the queue for the ping response, an error may be written to an error log. As well, if the message cannot be sent immediately, it may be queued for transmission whenever a connection becomes available.

[00104] For the second type of "ping", i.e. request for device statistics, if there is no record for the current MobileID in the database table at the middleware server 44, one may be inserted and initialized with the current time (e.g. in UTC) in the dtmLastStatisticsRequest field. The rest of the fields may be null. If there is a record for the current MobileID, the dtmLastStatisticsRequest field may be set to the current time, and the dtmLastStatisticsResponse and varStatisticsResponse fields may be set to null. The other fields may remain unaltered.

[00105] Following the table update, a message with the format 1500 illustrated in FIG. 15 may be enqueued for transmission. The priority of this message may be set to high, so that it is queued ahead of any existing messages. The message may then be received by the device through normal device push or pull, depending on how the mobile device is configured. Also, if the record in TBLMOBILEREQUESTS cannot be created/updated, it may be desirable to notify the administrator and to abort the process without queueing the message for transmission. Similarly, if the message cannot be queued for any reason, it may be desired to notify the administrator.

[00106] When the ping response is received from the mobile device, the TBLMOBILEREQUESTS database table may be updated for the responding MobileID. The field dtmLastStatisticsResponse may be set to the current date and time, and varStatisticsResponse may be filled with the text of the <PINGRESP> tag. If the record in TBLMOBILEREQUESTS cannot be updated, an error may be

written to the error log.

[00107] The ping response may be viewable by the system administrator from using the middleware server software 68. The dtmLastStatisticsRequest, dtmLastStatisticsResponse and varStatisticsResponse fields may be read from the table TBLMOBILEREQUESTS. A screen may be displayed (e.g. may pop up), containing these two date/times, formatted using the current time zone and locale information. The screen may contain a scrollable memo box for displaying the response text and a "Refresh" button, which will retrieve the current values from the table.

[00108] As before, if there is no record for the current mobile device, or it cannot be retrieved, the fields may be displayed as empty. It is noted that administrator notification of the receipt of a ping response does not necessarily occur automatically. The system administrator may be required to manually refresh the screen until the ping response is retrieved. However, the administrator may be able to watch an outgoing queue for that device to determine when the device receives the ping.

[00109] At the mobile device side, when a statistics "ping" request has been received, the device immediately formats, queues, and sends a statistics response, including the following statistics retrieved from the device:

Virtual Machine (also referred to as "Smart Client") Configuration
Send/Receive Interval
Other Intervals
Server Address
Notifications

Virtual machine Diagnostics
Virtual Machine Version
List of last 10 errors that occurred on the device
Top 3 messages in the transaction queue (text)
Number of queued messages

Registered Application ID's
Currently open screen

Device Information
Make & Model
Battery Remaining
Memory Free
Current Network
PIN Number
Operating System Version
Current Date/Time

[00110] It will be appreciated that the set of statistics that is sent may be different from the set of statistics described above in alternative embodiments. The response priority may be set to high, so that the response will be sent ahead of any other messages.

[00111] The statistics response may have the format 1600 of FIG. 16. Depending on the nature of the reported statistics, it may be desired to pass the <PINGRESP> through an XML formatter to replace any illegal XML characters.

[00112] The retrieval of the requested statistics may be performed in the following way on the devices:

Smart Client (i.e. virtual machine) Configuration
Send/Receive Interval – from tblOptions
Other Intervals – from tblOptions
Server Address – from tblOptions
Notifications – from tblOptions

Smart Client Diagnostics
Smart Client Version – from constant ASC_VERSION
List of last 10 errors that occurred on the device – stored in table tblErrorLog
Top 3 messages in the transaction queue (text) – from tblMessageQueue

Number of queued messages – count records in tblMessageQueue

Registered Application ID's – count registered applications from tblApplications

Currently open screen – use Name property of current instance of AIRIXScreen

Device Information (RIM)

Make & Model – RadioInfo.getNetworkType() – NETWORK_GPRS (7200/6710/6200/5810), NETWORK_CDMA (6750), NETWORK_IDEN (6510)

Battery Remaining – DeviceInfo.getBatteryLevel()

Memory Free – MemoryStats.getFree()

Current Network – RadioInfo.getNetworkType()

PIN Number – DeviceInfo.getDeviceId()

Operating System Version – DeviceInfo.getOSVersion()

Current Date/Time – Calendar.getInstance()

Device Information (PPC)

Make & Model – SystemParamatersInfo(SPI_GETOEMINFO)

Battery Remaining – GetSystemPowerStatusEx()

Memory Free – GetDiskFreeSpaceEx()

Current Network – vendor specific

PIN Number – vendor specific

Operating System Version – SystemParametersInfo(SPI_GETPLATFORMTYPE)

Current Date/Time – GetLocalTime()

Device Information (Palm)

To be determined

[00113] If no entry can be created in the queue for the device statistics response, an error may be written to an error log. As well, if the message cannot be sent immediately, it may be queued for transmission whenever a connection becomes available.

[00114] It will be understood that the invention is not limited to the embodiments described herein which are merely illustrative of a preferred embodiment of carrying out the invention, and which is susceptible to modification of form, arrangement of parts, steps, details and order of operation. The invention, rather, is intended to encompass all such modification within its scope, as defined by the claims.